DOM (“Document Object Model”)

Porfírio Filipe
Paulo Trigo Silva

Tópicos

- Descrição comparativa
  - Document Object Model (DOM)
  - Simple API for XML (SAX)
- Descrição da interface de programação do DOM
- Exemplos de utilização

Porfírio Filipe
Paulo Trigo Silva

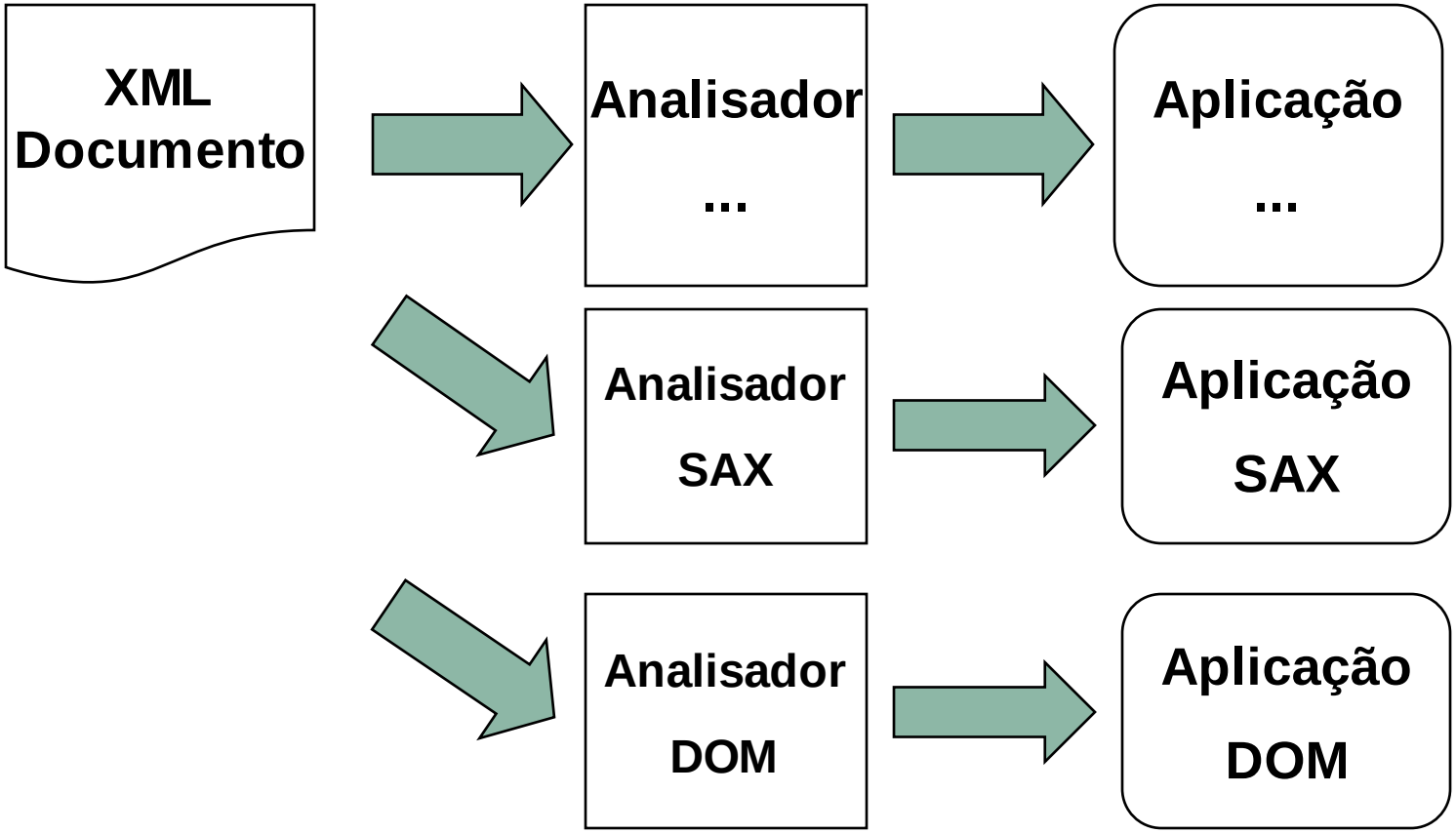
Processamento XML

- Baseado em texto
  - documentos XML são baseados em texto $\Rightarrow$ os processadores podem analisá-los sem recorrer a qualquer outro instrumento;
  - ... tratamento pouco produtivo e sujeito a erros; no entanto, é realizado pelas actuais ferramentas (analisadores, editores, etc)
- Baseado em eventos
  - analisador (e.g. SAX) gera eventos (e.g. início/fim documento, início/fim elemento, etc) durante análise do documento
- Baseado em árvore
  - analisador (e.g. DOM) gera a árvore que representa o documento
  - ... sobre a árvore é permitido percorrer, retroceder, aceder e alterar os elementos, inserir novos elementos, etc.

Porfírio Filipe
Paulo Trigo Silva

Aplicação XML

- Transformação de um documento XML num formato que possa ser facilmente manipulado por um programa



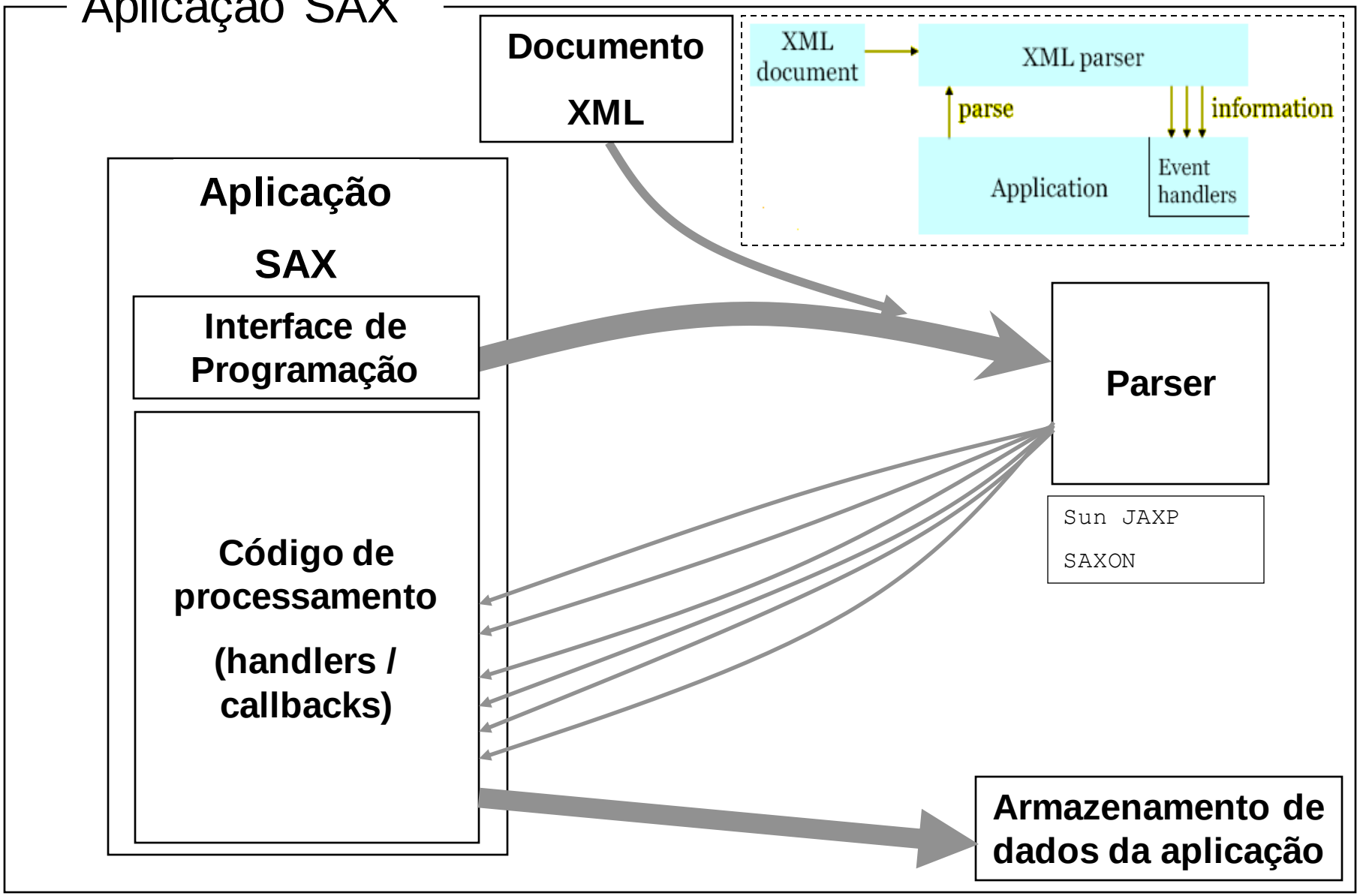
Porfírio Filipe
Paulo Trigo Silva

O que é o SAX

- *Simple API for XML*
  - É um *standard* “de facto”
  - O documento XML é tratado como um fluxo unidireccional de caracteres
  - O modelo de programação é baseado em eventos (“event-based API”)
  - A análise do documento XML origina a chamada de procedimentos
- Vantagens
  - Exige pouca memória
  - Não precisa de ler o documento inteiro
  - Indica a ocorrência de erros de forma precisa
- Desvantagens
  - Acesso sequencial “*Sequential access*” para leitura
  - Obriga o programador a manter registo do contexto
  - Não permite actualizar o documento XML
  - Não permite a criação de documentos XML

Porfírio Filipe
Paulo Trigo Silva

Aplicação SAX



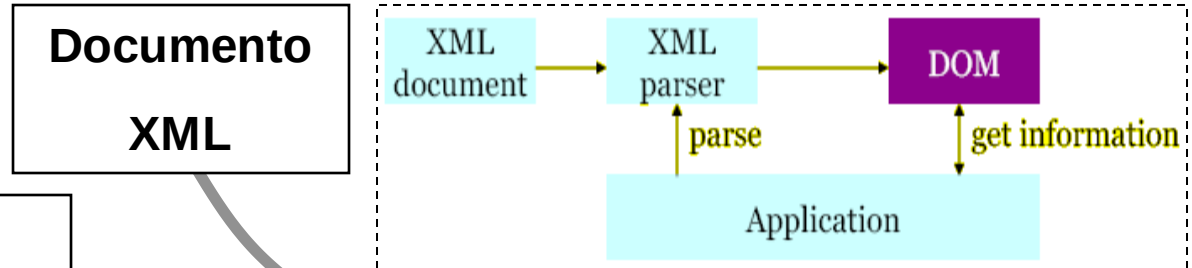
Porfírio Filipe
Paulo Trigo Silva

O que é o DOM

- *Document Object Model*
  - É uma recomendação do W3C
  - O documento XML é integralmente armazenado em memória
  - O modelo de programação é baseado numa árvore “Tree-based API”
- Vantagens
  - Armazenamento dos dados em memória
  - Acesso aleatório “Random access”
  - Navegação multi-direccional
  - Permite leitura e escrita
  - Permite actualizar o documento XML
- Desvantagens
  - Necessidade uma quantidade apreciável de memória
  - Requer o carregamento integral do documento
  - A existência de erros no documento XML impede o seu carregamento
  - Não pode ser usado em documentos XML de grandes dimensões

Porfírio Filipe
Paulo Trigo Silva

Aplicação DOM



Documento XML

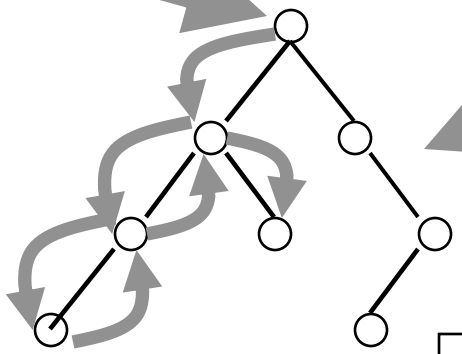
Aplicação DOM

Interface de Programação

Código de processamento

Parser

Árvore



Sun JAXP
IBM XML4J
Apache Xerces

Armazenamento de dados da aplicação

Porfírio Filipe
Paulo Trigo Silva

Níveis DOM

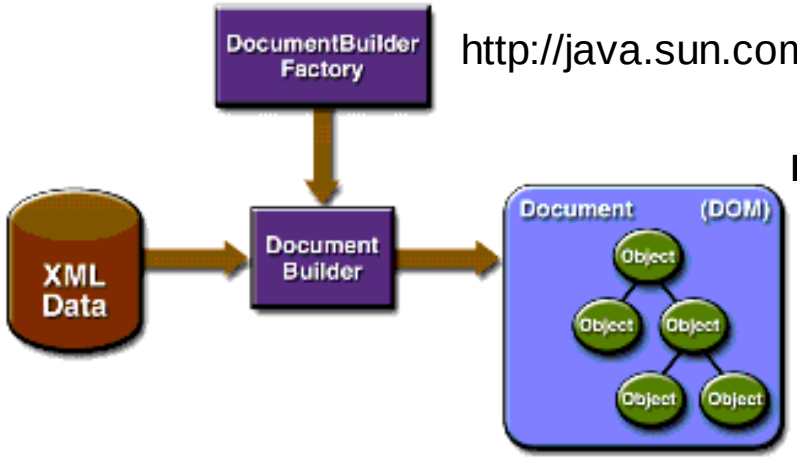
- Nível 0 - Primeira recomendação do W3C
  - permite aos *browsers* identificar e manipular elementos numa página
- Nível 1 - Inclui suporte para XML e HTML
  - `http://www.w3.org/TR/REC-DOM-Level-1`
- Nível 2 - Permite o uso de *namespace*
  - API mais sofisticada com eventos e CSS
  - `http://www.w3.org/TR/DOM-Level-2`
- Nível 3 - Suporte avançado a *namespaces*,
  - suporta eventos, DTD, XML Schema, XPATH e XSLT
  - `http://www.w3.org/TR/DOM-Level-3`

Porfírio Filipe
Paulo Trigo Silva

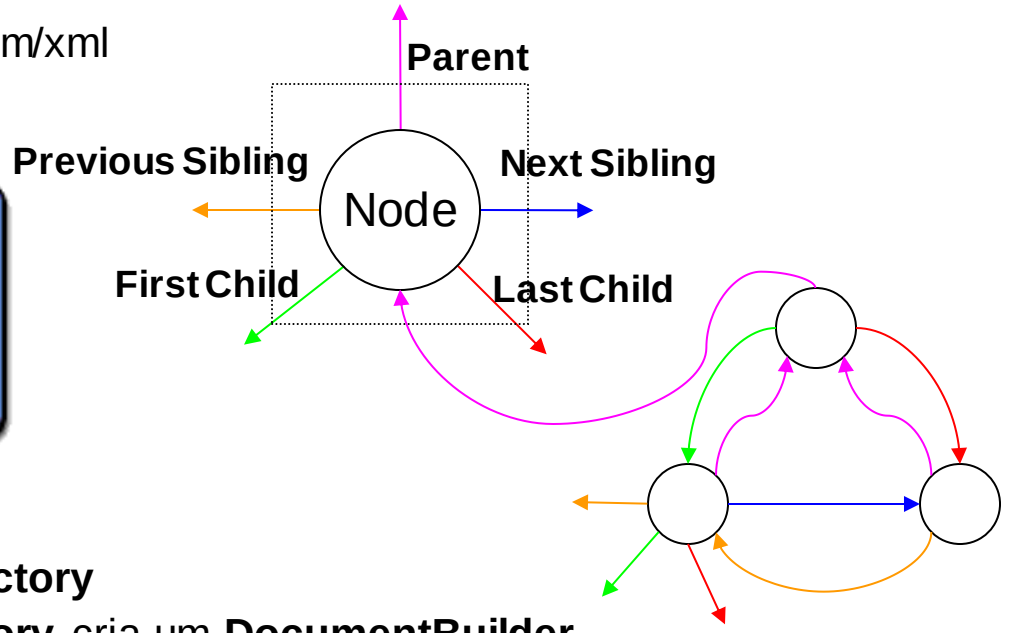
Utilização do DOM

Java API para XML *Parsing* (JAXP)

- **org.w3c.dom** – Define as interfaces de programação para XML.
- **javax.xml.parsers** - Essencialmente, define as interfaces/classes **DocumentBuilderFactory**, **DocumentBuilder** e **Document**.



http://java.sun.com/xml



- Utilização
  - Criar um **DocumentBuilderFactory**
  - Usar o **DocumentBuilderFactory** cria um **DocumentBuilder**
  - O **DocumentBuilder** faz a análise e cria o **Document**
  - O **Document** é utilizado para manipular aos nós da árvore

Porfírio Filipe
Paulo Trigo Silva

Aplicação DOM "*Hello World*"

```
<?xml version="1.0" ?>
<display>Hello World!</display>
```

hello.xml

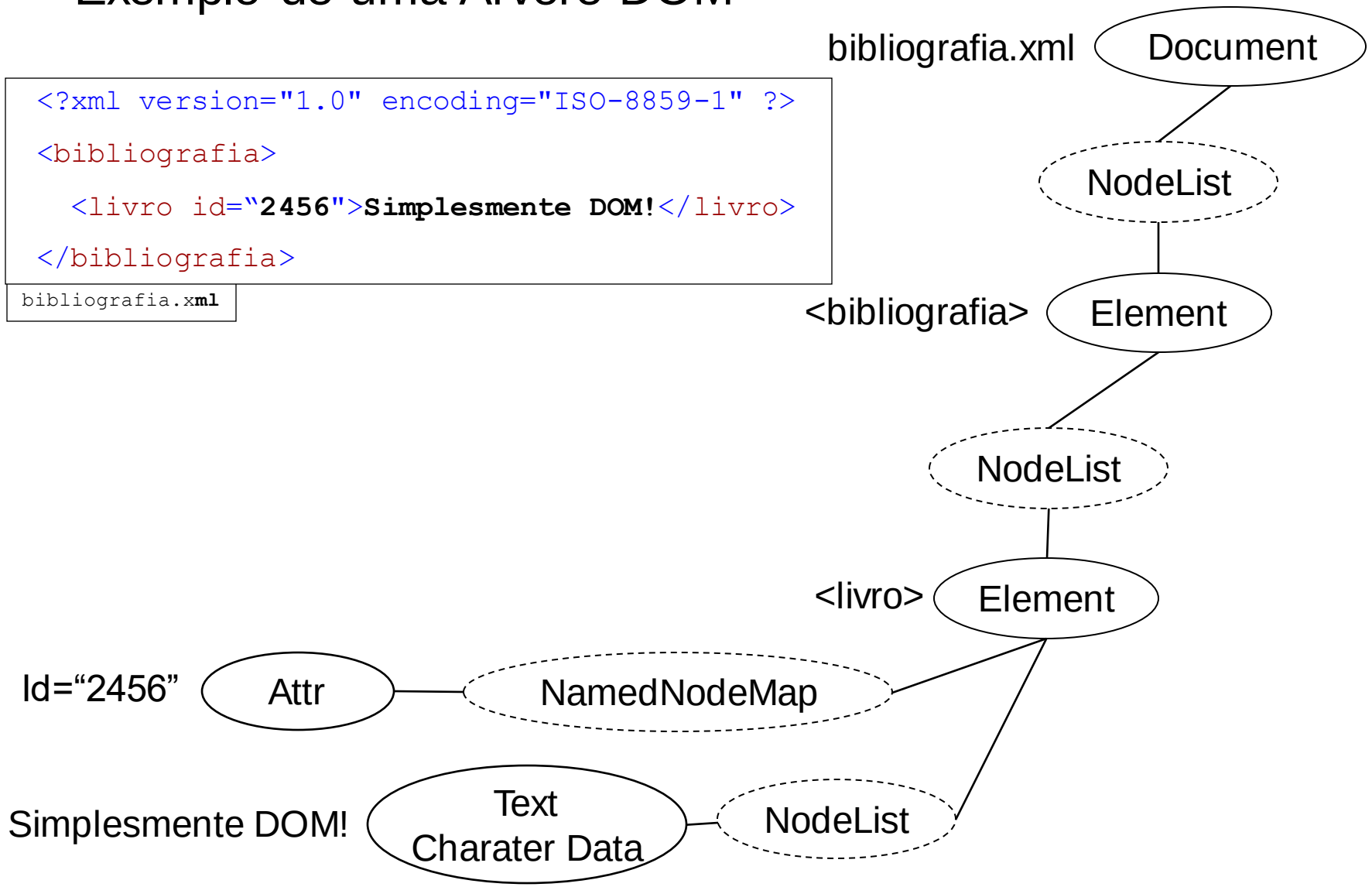
```
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class DomApp {
  public static void main(String args[]) {
    try {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse("hello.xml");
        Element root = document.getDocumentElement();
        Node textNode = root.getFirstChild();
        System.out.println( textNode.getNodeValue() );
      } catch (Exception e) {
              e.printStackTrace(System.out);
             System.out.print("Erro ao analisar o documento XML.");
  }}}
```

Porfírio Filipe
Paulo Trigo Silva

Exemplo de uma Árvore DOM

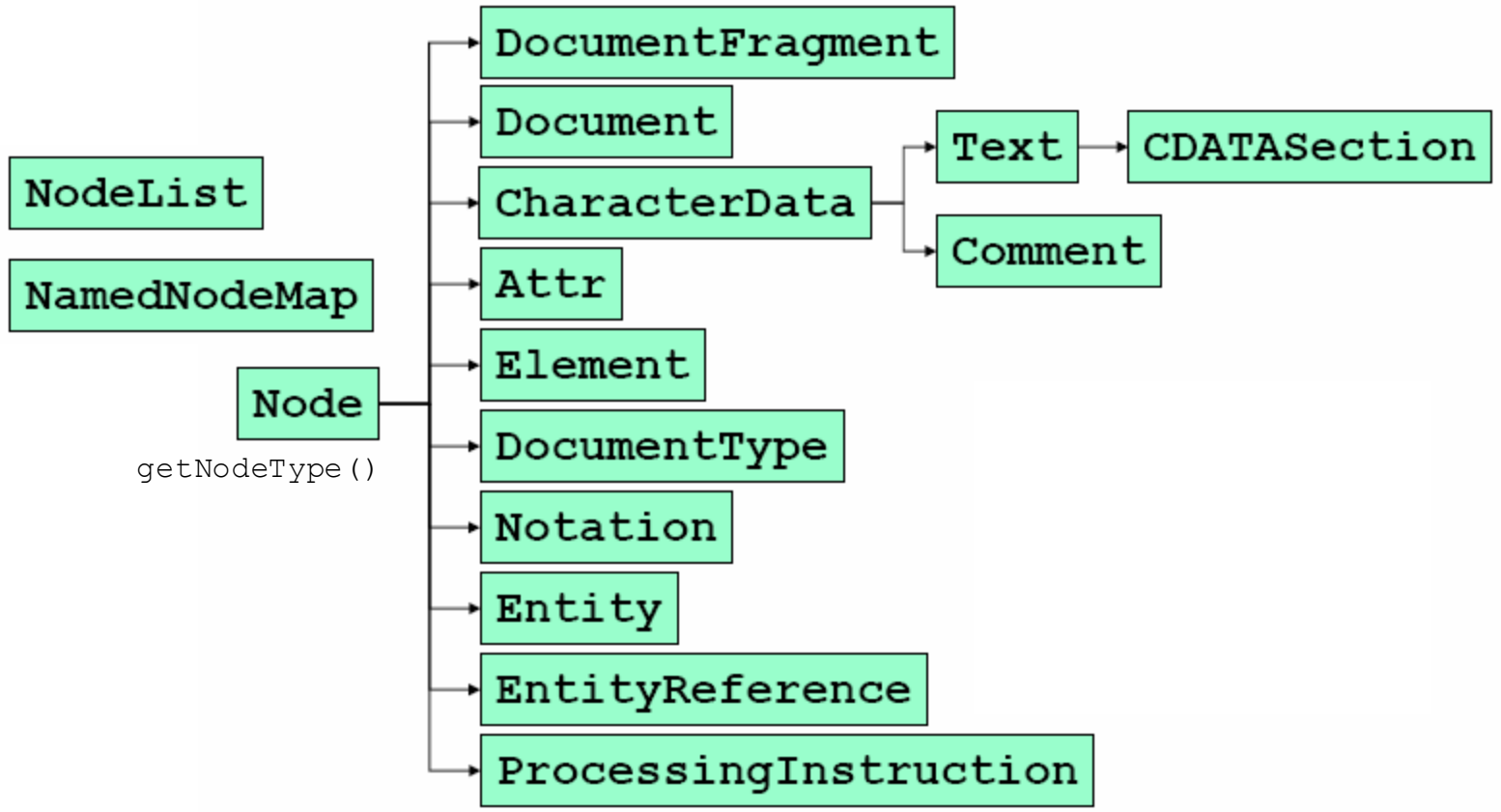
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<bibliografia>
  <livro id="2456">Simplesmente DOM!</livro>
</bibliografia>
```

bibliografia.xml



Porfírio Filipe
Paulo Trigo Silva

Interfaces/classes DOM



Nota: todos os nós derivam de *org.w3c.dom.Node*

Porfírio Filipe
Paulo Trigo Silva

Tipos de Nós mais Comuns

***node*.`getNodeType()`**	**Exemplo: "como será o nó?"**
ELEMENT_NODE	<**artista** categoria="banda"> The Beatles </**artista**>
ATTRIBUTE_NODE	**categoria="banda"**
TEXT_NODE	**The Beatles**
PROCESSING_INSTRUCTION_NODE	<**?xml version**="1.0"?>
DOCUMENT_TYPE_NODE	<**!DOCTYPE discos SYSTEM "discos.dtd"**>
COMMENT_NODE	**<!-- o meu comentário -->**

Porfírio Filipe
Paulo Trigo Silva

Node

- Grupos de métodos
  - Básico
    - ◊ Nome
    - ◊ Tipo
    - ◊ Conteúdo
  - Navegação
    - ◊ Pai
    - ◊ Filho
    - ◊ Irmão
  - Manipulação
    - ◊ Alteração
    - ◊ Reorganização

Porfírio Filipe
Paulo Trigo Silva

Node

- Métodos básicos aplicáveis a um nó:
  - short getNodeType()
    - ◊ e.g. Node.ELEMENT_NODE
  - String getNodeName()
  - String getNodeValue()
  - void setNodeValue(String value)

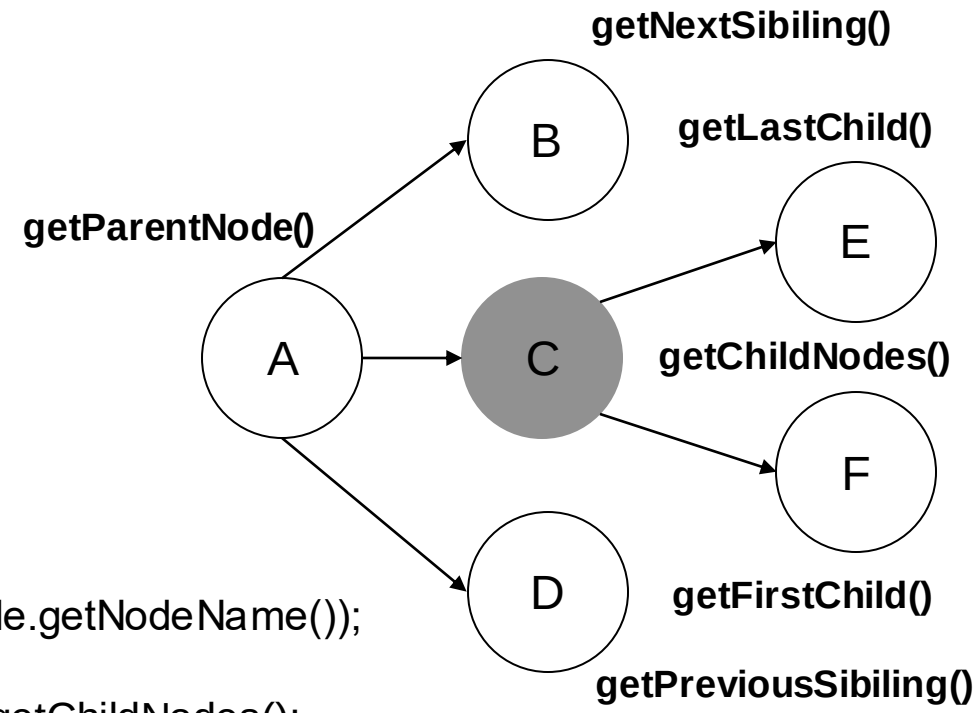
é valor constante!

Método / Tipo Nó	**`Element`**	**`Text`**	**`Attr`**
getNodeName()	*nome da marca*	`"#text"`	*nome do atributo*
getNodeValue()	`null`	*texto*	*valor do atributo*
getNodeType()	ELEMENT_NODE	TEXT_NODE	ATTRIBUTE_NODE

Porfírio Filipe
Paulo Trigo Silva

Node Navegação

- Tipos de nó que podem ter filhos
  - Document, DocumentFragment e Element
- Seleccionar filhos, pais e irmãos
  - Node getFirstChild()
  - Node getLastChild()
  - Node getNextSibling()
  - Node getPreviousSibling()
  - Node getParentNode()
  - NodeList getChildNodes()
- Listar o nome de todos os nós

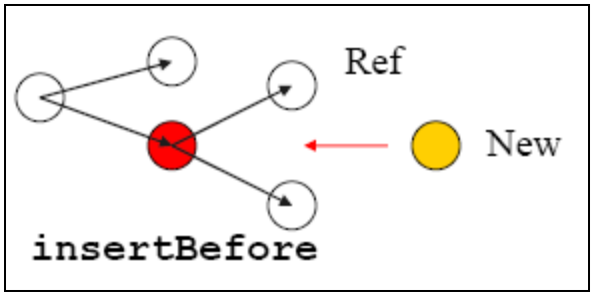


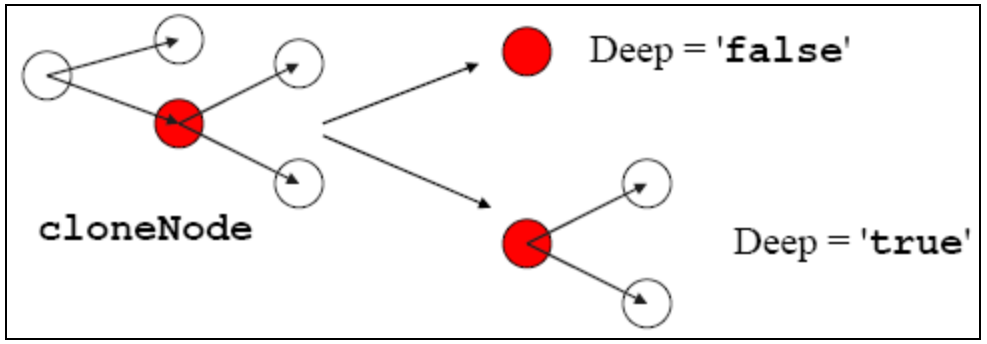

```
public void traverse(Node node) {
      System.out.println("node: "+node.getNodeName());
      if (node.hasChildNodes()) {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++) {
               Node child = children.item(i);
               traverse(child);
      } } }
```

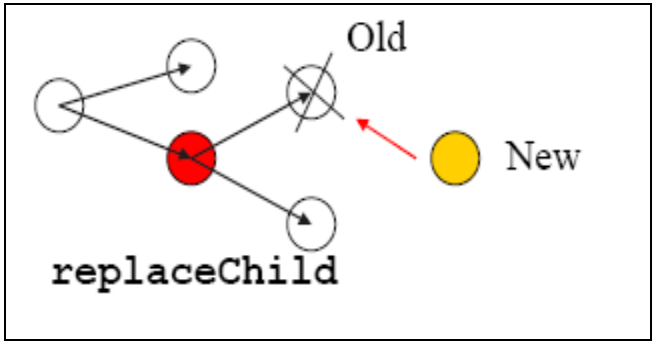
Porfírio Filipe
Paulo Trigo Silva

Node Manipulação

- Os filhos de um nó podem ser adicionados, removidos, substituidos, copiados, movidos, etc.
  - Node removeChild(Node old)
  - Node insertBefore(Node new, Node ref)
  - Node appendChild(Node new)
  - Node replaceChild(Node new, Node old)
  - Node cloneNode(boolean deep)







Porfírio Filipe
Paulo Trigo Silva

Pecurso numa Árvore DOM

- Listar os nomes dos nós da árvore

```
public void traverse(Node node) {
        System.out.println("node: "+node.getNodeName());
        if (node.hasChildNodes()) {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++) {
               Node child = children.item(i);
               traverse(child);
        }}}
```

- Navegar nos nós referindo marcas

```
public void iterateBibliografia(Document document) {
      NodeList livros = document.getElementsByTagName("livro");
      int length = livros.getLength();
      for (int i = 0; i < length; i++) {
            Node livro = livros.item(i);
            // fazer o que for necessário com o livro
   }}
```

Porfírio Filipe
Paulo Trigo Silva

Document

- A raíz da árvore DOM representa todo o documento XML
  - DocumentType getDocumentType() // Informação sobre o DOCTYPE - Ver 'DocumentType'
  - DOMImplementation getImplementation() // Informação sobre as capacidades da implementação DOM
  - Element getDocumentElement() // Retorna referencia para o nó raíz
  - NodeList getElementsByName(String tagName) // Retorna os elementos com ocorrências da marca 'tagName'
- Criar nós
  - Element createElement(String tagName)
  - Text createTextNode(String data)
  - Comment createComment(String data)
  - CDATASection createCDATASection(String data)
  - ProcessingInstruction createProcessingInstruction(String target, String data)
  - Attr createAttribute(String name)

Porfírio Filipe
Paulo Trigo Silva

Element

- Básico
  - String getTagName()
  - NodeList getElementsByTagName(String tagName)
  - void normalize() // junção de elementos de texto
  - String getTextContent() // devolve o texto do elemento
  - void setTextContent(String value) // afeta o texto do elemento
- Manipular atributos
  - String getAttribute(String name)
  - void setAttribute(String name, String value)
  - void removeAttribute(String name)
  - Attr getAttributeNode(String name)
  - void setAttributeNode(Attr new)
  - void removeAttributeNode(Attr old)
- Obter lista de atributos
  - NamedNodeMap getAttributes()

Porfírio Filipe
Paulo Trigo Silva

Attr

- Interface para tratar atributos
  - String getName() // retorna o nome do atributo
  - String getValue() // retorna o valor do atributo
  - void setValue(String value) // modifica o valor do atributo
  - boolean getSpecified()   // false – se indicado no DTD

- Exemplo de criação de atributos

```
Attr newAttr = myDoc.createAttribute("estado"); //cria nó vazio
newAttr.setValue("vazio");                      //afecta o valor do atributo
myElement.setAttributeNode(newAttr) // liga o atributo no elemento
```

Porfírio Filipe
Paulo Trigo Silva

Text e Comment

- Representa o conteúdo textual dos nós *Element*
  - São sempre nós folha
  - Único método próprio (não herdado)
    - ◊ Text splitText(int offset) // Quebra texto em dois
- O método normalize(), aplicado a um Nó, concatena os objectos *Text* das folhas a partir deste nó
- Podem ser criados a partir da interface *Document*
- Exemplo de criação do comentário '<!-- o meu comentário -->'

```
Comment nComenta = myDoc.createComment("o meu comentário")
```

Porfírio Filipe
Paulo Trigo Silva

NodeList

- Representa uma colecção ordenada de Nós
  - int getLength() // quantidade de nós existentes
  - Node item(int index) // retorna o Nó da posição “index”

- Percorrendo os nós filho de um elemento

```
Node child;
NodeList children = elemento.getChildNodes();

for (int i = 0; i < children.getLength(); i++) {
    child = children.item(i);
    if (child.getNodeType() == Node.ELEMENT_NODE)
            System.out.println(child.getNodeName());
}
```

Porfírio Filipe
Paulo Trigo Silva

NamedNodeMap

- Colecção não ordenada de Nós (e.g. Attribute, Entity e Notation)
  - int getLength()
  - Node item(int index)
  - Node getNamedItem(String name)
  - Node setNamedItem(Node node)
  - Node removeNamedItem(String name)
- Acesso depende de nome único
- Exemplos de utilização

```
NamedNodeMap myAttributes = myElement.getAttributes();
NamedNodeMap myEntities = myDocument.getEntities();
NamedNodeMap myNotations = myDocument.getNotations();
```

Porfírio Filipe
Paulo Trigo Silva

Utilização no Bowser

```
<!DOCTYPE html>
<html> <body>
<p id="demo"></p>
<script>
    var parser, xmlDoc;
    var text = "<bookstore><book>" +
    "<title>The Definitive Guide to JSF in Java EE 8</title>" +
    "<author>Bauke Scholtz</author>" +
    "<author>Arjan Tijms</author>" +
    "<year>2018</year>" +
    "</book></bookstore>";
    parser = new DOMParser();
    xmlDoc = parser.parseFromString(text,"text/xml");
    document.getElementById("demo").innerHTML =
    xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body></html>
```

Porfírio Filipe
Paulo Trigo Silva

Javascript – Principais Propriedades de *Element*

Property	Description
attributes	**Returns a NamedNodeMap that contains all attributes of a node**
childNodes	**Returns a node list that contains all children of a node**
firstChild	**Returns the first child node of a node**
lastChild	**Returns the last child node of a node**
nextSibling	**Returns the node immediately following a node. Two nodes are siblings if they have the same parent node**
nodeName	**Returns the name of the node (depending on the node type)**
nodeType	**Returns the node type as a number**
nodeValue	**Returns the value of the node**
ownerDocument	**Returns the Document object of a node (returns the root node of the document)**
parentNode	**Returns the parent node of a node**
previousSibling	**Returns the node immediately preceding a node. Two nodes are siblings if they have the same parent node**
tagName	**Returns the name of the element node**

Porfírio Filipe
Paulo Trigo Silva

Javascript – Principais Métodos de *Element*

Method	Description
appendChild(newnode)	**Appends a new child node to a node**
cloneNode(boolean)	**Creates an exact clone node of a node. If the boolean parameter is set to true, the cloned node clones all the child nodes of the original node as well**
getAttribute(attrname)	**Returns the value of the specified attribute**
AttributeNode(attrname)	**Returns the specified attribute node as an Attr object**
getElementsByTagName(tagname)	**Returns the specified node, and all its child nodes, as a node list**
hasChildNodes()	**Returns true if a node has child nodes. Otherwise it returns false**
insertBefore(newnode,refnode)	**Inserts a new node (newnode) before the existing node**
(refnode)normalize()	**Combines all subtree Text nodes into a single one**
removeAttribute(attrname)	**Removes the specified attribute's value**
removeAttributeNode(attriname)	**Removes the specified attribute node**
removeChild(nodename)	**Removes the specified node and returns it**
replaceChild(newnode,oldnode)	**Replaces the oldnode with the newnode, and returns the old node**
setAttribute(attrname,attrvalue)	**Sets the value of the named attribute**
setAttributeNode(attrname)	**Inserts the specified new attribute to the element**

Porfírio Filipe
Paulo Trigo Silva